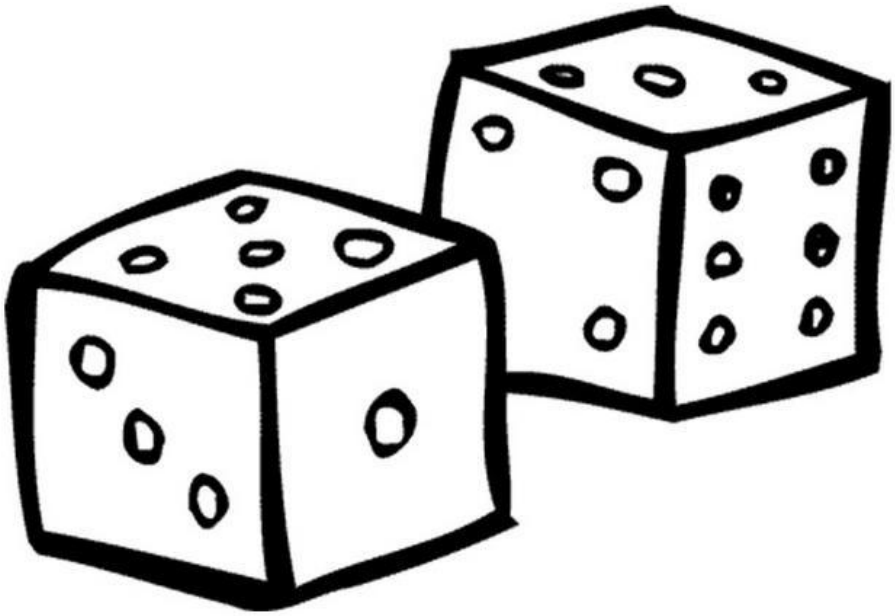
Nancy-Université
Université Nancy 2

IGA

Rapport

Projet Design Pattern



Réalisé par :

Abdelaziz Mahamat Ali

Issam Fadil

Encadré par :

Mr Ben ali

Année universitaire : 2007/2008

Sommaire

Introduction

Les Design Patterns (en français Patrons de conception, Modèles de conception ou encore Motifs de conception sont un recueil de bonnes pratiques de conception pour un certain nombre de problèmes récurrents en programmation orientée objet.

Les designs patterns décrivent des organisations pratiques de classes objets. Ces organisations résultent souvent d'une conception empirique, le concepteur objet tente de faciliter la réutilisation et la maintenance du code. On peut donc concevoir un modèle d'application comme une forme d'organisation transposable à plusieurs applications. Ces systèmes peuvent apparaître complexes aux débutants voire inutiles, il est pourtant très important d'en connaître plusieurs et de les appliquer systématiquement (dans les cas reconnus comme pouvant évoluer).

Les patrons de conception décrivent des solutions standard pour répondre à des problèmes d'architecture et de conception des logiciels. À la différence d'un algorithme qui s'attache à décrire d'une manière formelle comment résoudre un problème particulier, les patrons de conception décrivent des procédés de conception généraux. On peut considérer un patron de conception comme une formalisation de bonnes pratiques.

On peut donc considérer les patrons de conception comme un outil de capitalisation de l'expérience appliqué à la conception logicielle.

Le but général des patrons de conception est de minimiser les interactions qu'il peut y avoir entre les différentes classes (ou modules, plus généralement) d'un même programme. L'avantage de ces patrons est de diminuer le temps nécessaire au développement d'un logiciel et d'augmenter la qualité du résultat, notamment en appliquant des solutions déjà existantes à des problèmes courants de conception**.**

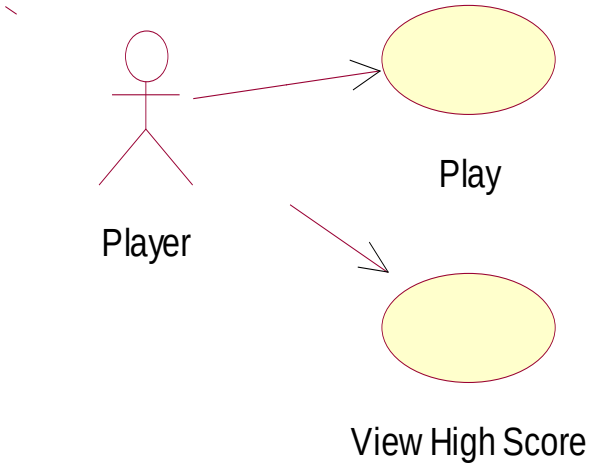
Description du projet

Le projet consiste à analyser et concevoir et implanter deux patrons de conceptions observateur, la fabrique abstraite en les appliquant sur un jeux de dès.

Le jeu consiste à lancer 10 fois 2 dés. Si le total des 2 dés fait 7, il marque 10 points à son score, sinon il n'en marque rien. En fin de partie, son score est inscrit dans le tableau des 'high scores'.

1 - L'analyse des besoins (définition des fonctionnalités attendues)

1-1 Diagramme de cas d'utilisation



Description du cas Play :

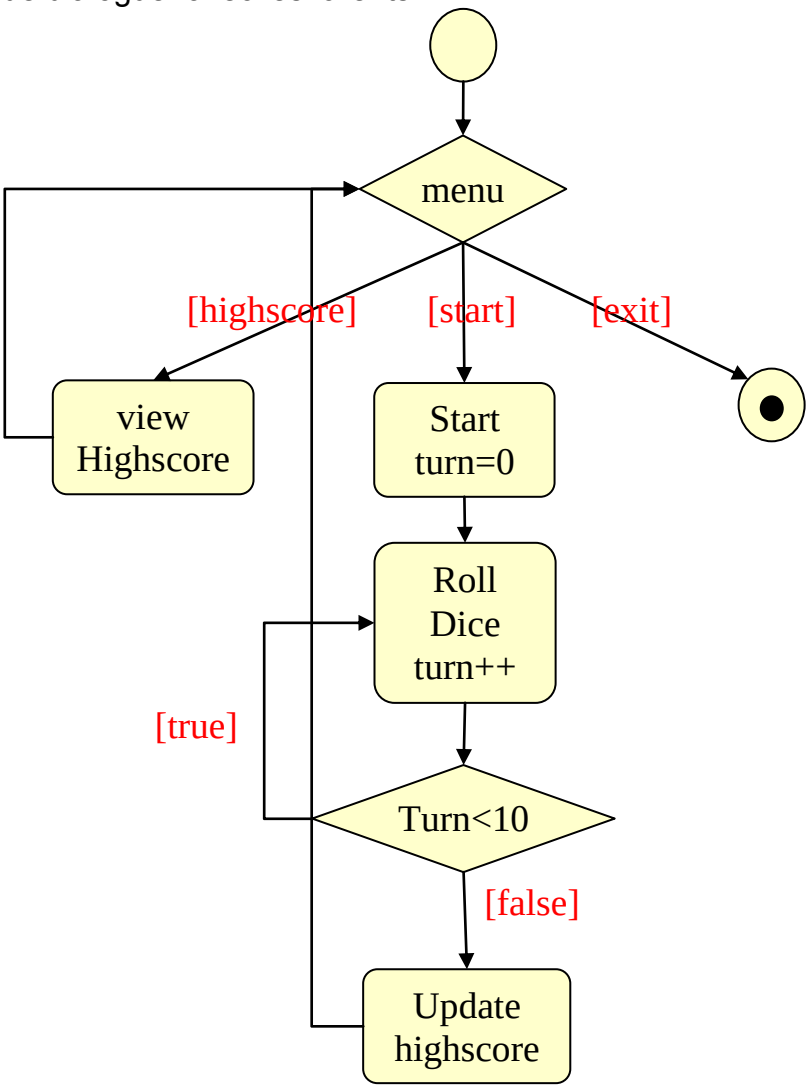
- Acteur : Player
- Description : Le joueur lance 10 fois les dés, à chaque fois que le total des deux dés fait 7, son score augmente de 10 points.

Description du cas View High Score :

- Acteur: Player
- Description: Le joueur consulte en lecture seul les high score

1-2 diagramme d'activité

Sa construction permet de décrire l'organisation générale des traitements et permet de dialoguer avec les 'clients'.

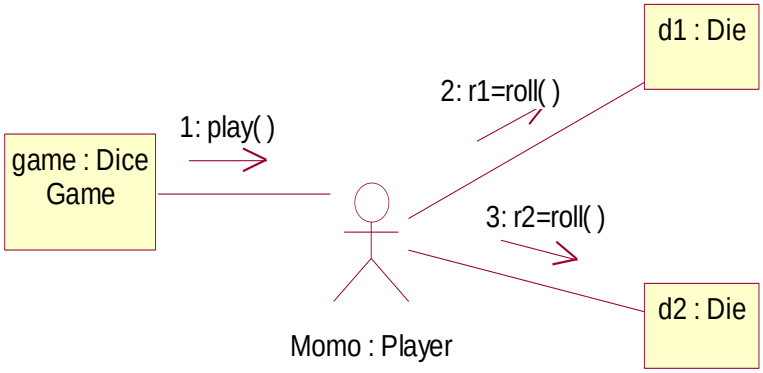


Il doit exister une certaine cohérence entre ces diagrammes. On peut vérifier que tous les cas se retrouvent quelque part dans le diagramme d'activités.

2. L’analyse (définition des classes du domaine et de leur dynamique)

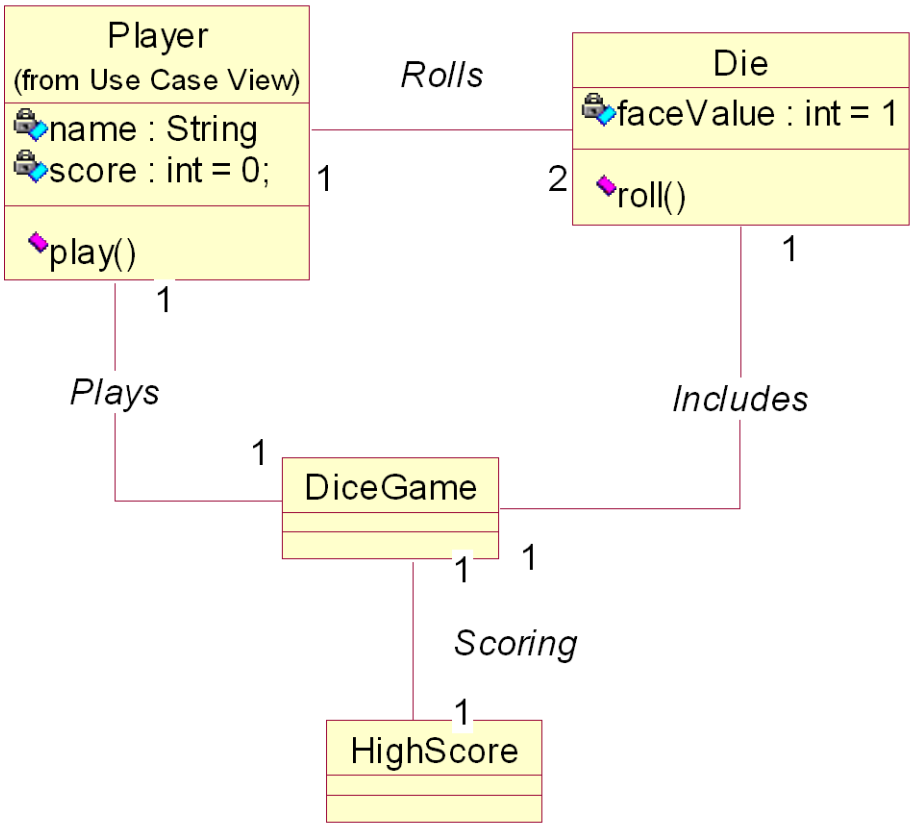
2-1 Diagramme de collaboration

Ce diagramme permet de visualiser des objets, leurs relations et l’ordonnancement des appels de messages.



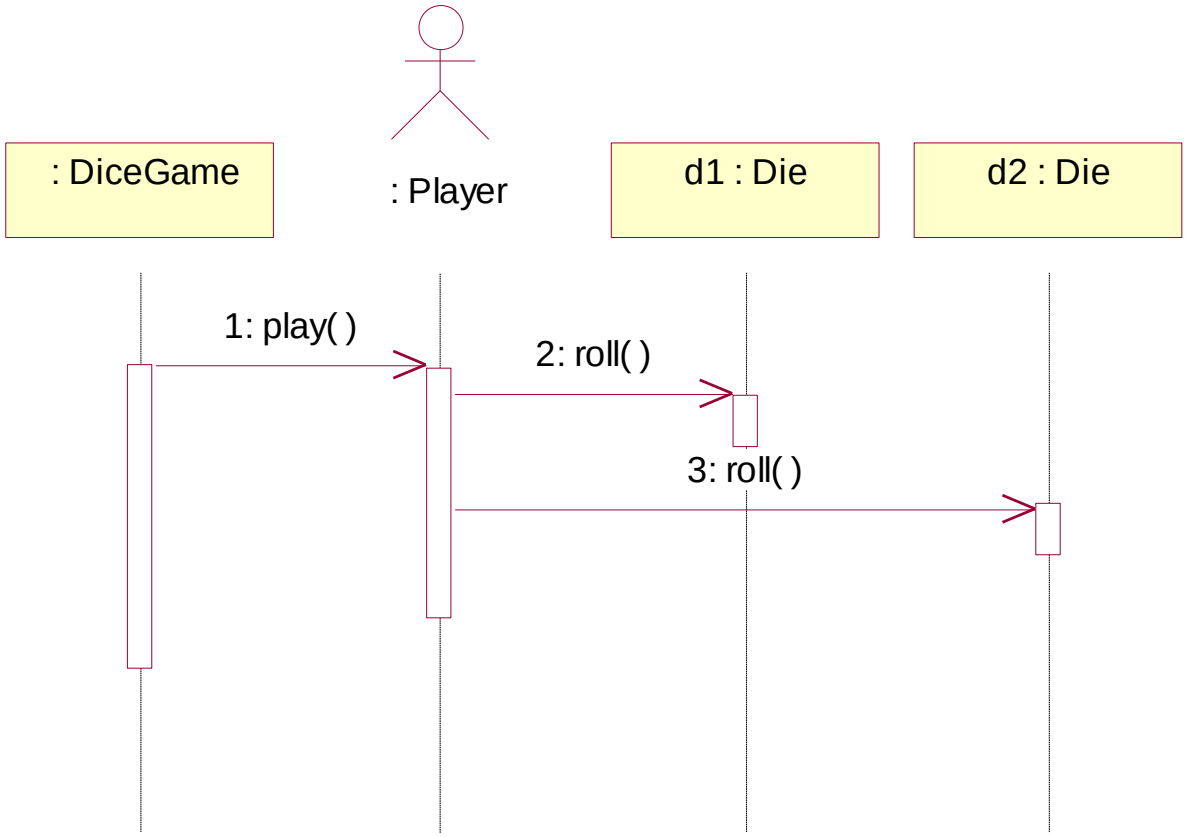
2-2 Diagramme de classes :

On complète les classes et on représente leurs associations statiques et dynamiques; on définit les attributs des classes et les cardinalités des associations.



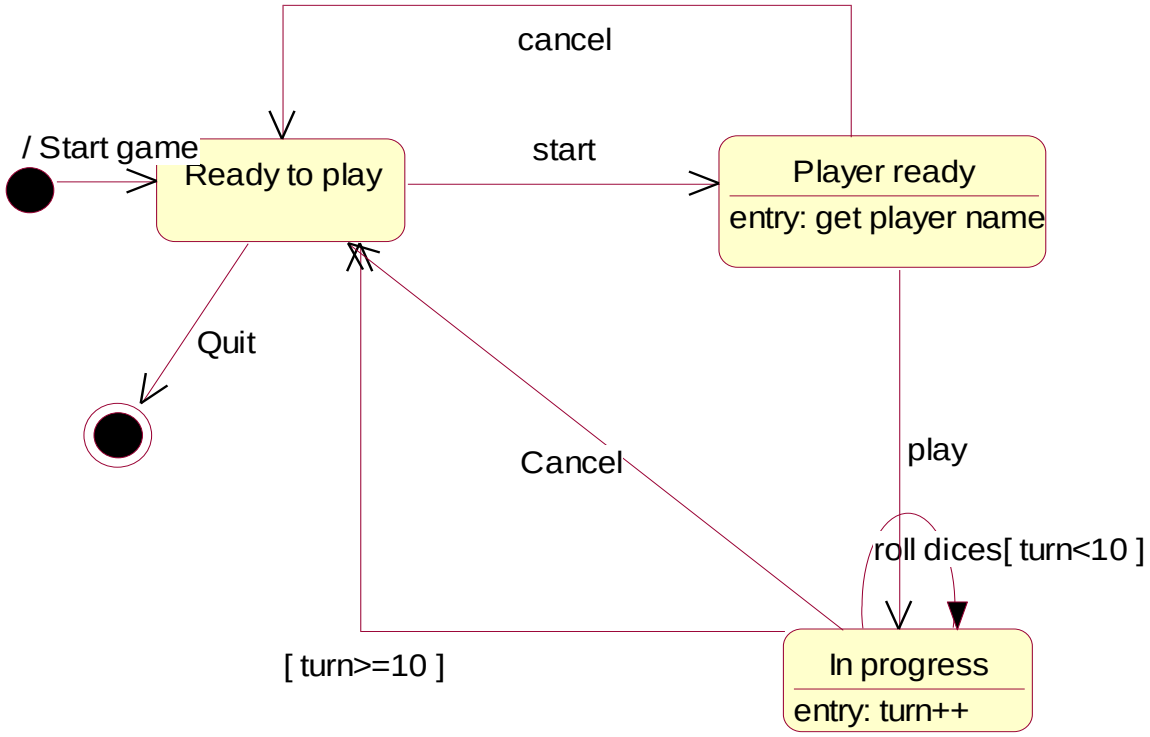
2-3 Diagramme de séquences

Ils modélisent la dynamique (comme diagrammes de collaboration) en se focalisant sur l'enchaînement des messages.



2-4 Diagramme d'état

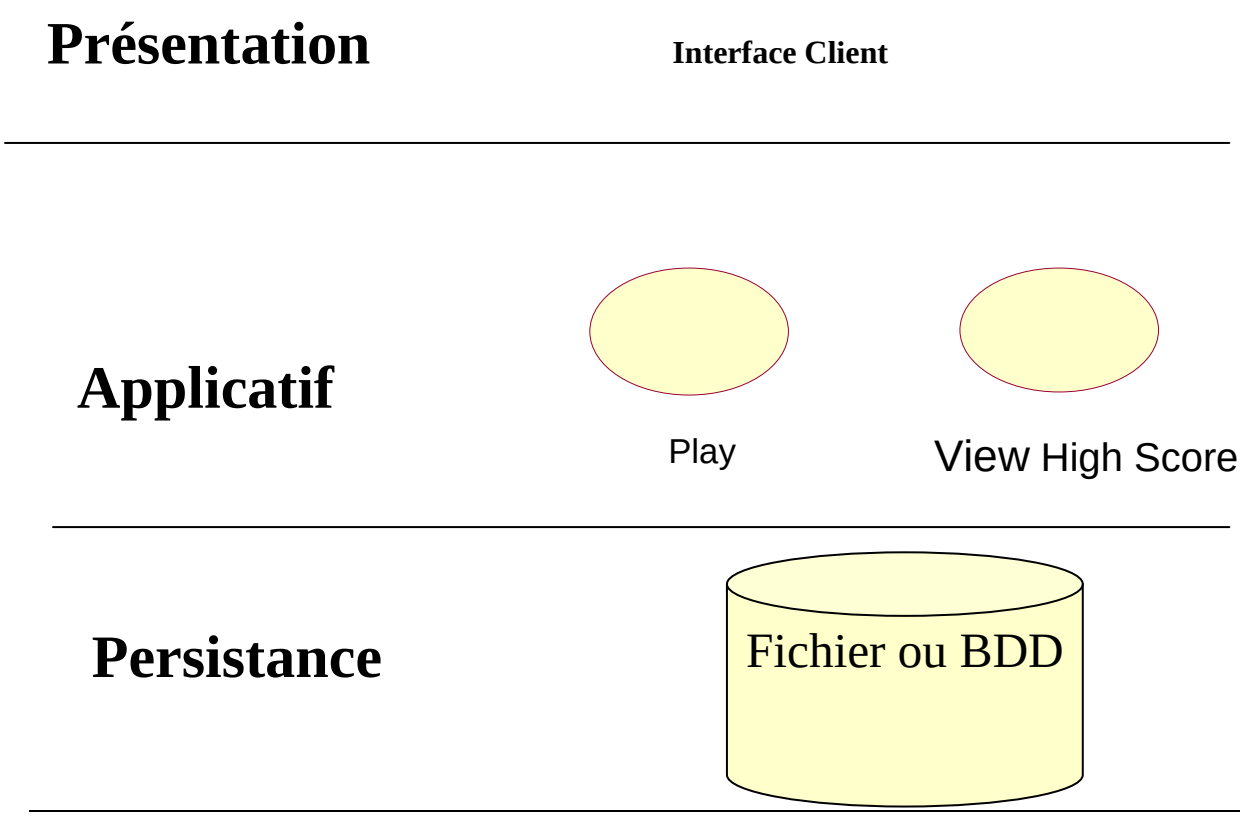
Il a pour objet de déterminer les états et transitions d'état d'un objet. Ici le diagramme d'états d'une partie (diceGame).



3. La conception

3-1 Architecture

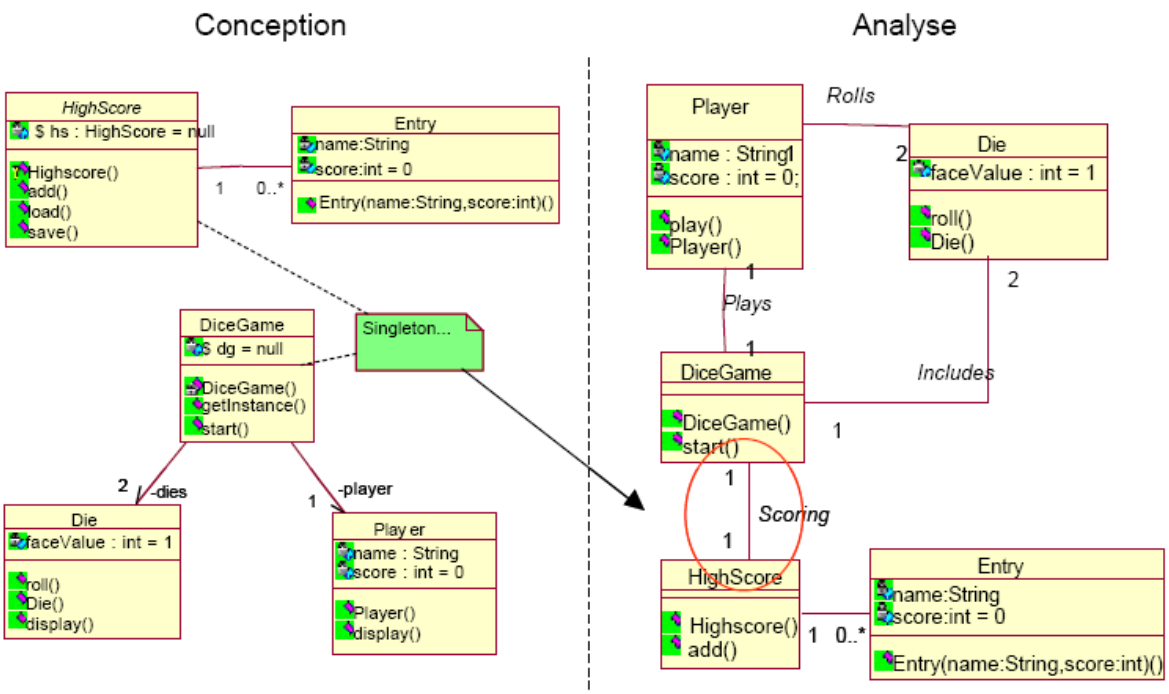
On fait le choix d'une architecture en couches comportant très classiquement 3 couches.



3-2 Conception du niveau applicatif

3-2-1 le pattern singleton

Assure qu'une classe à une instance unique qu'elle garde dans une variable de classe (static uniqueInstance) et donne un point d'accès depuis la classe à cette instance (méthode de classe static Instance() ou getInstance()).

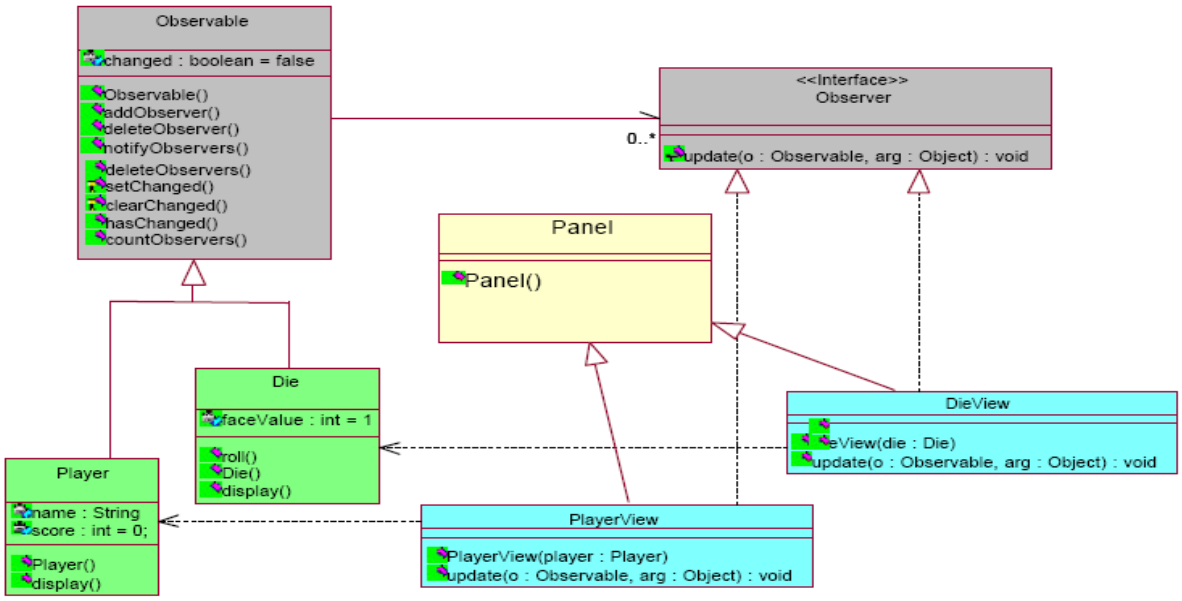


Sont également ajoutées des méthodes pour sauvegarder les high score (load, save) et pour afficher les dés et joueurs (display).

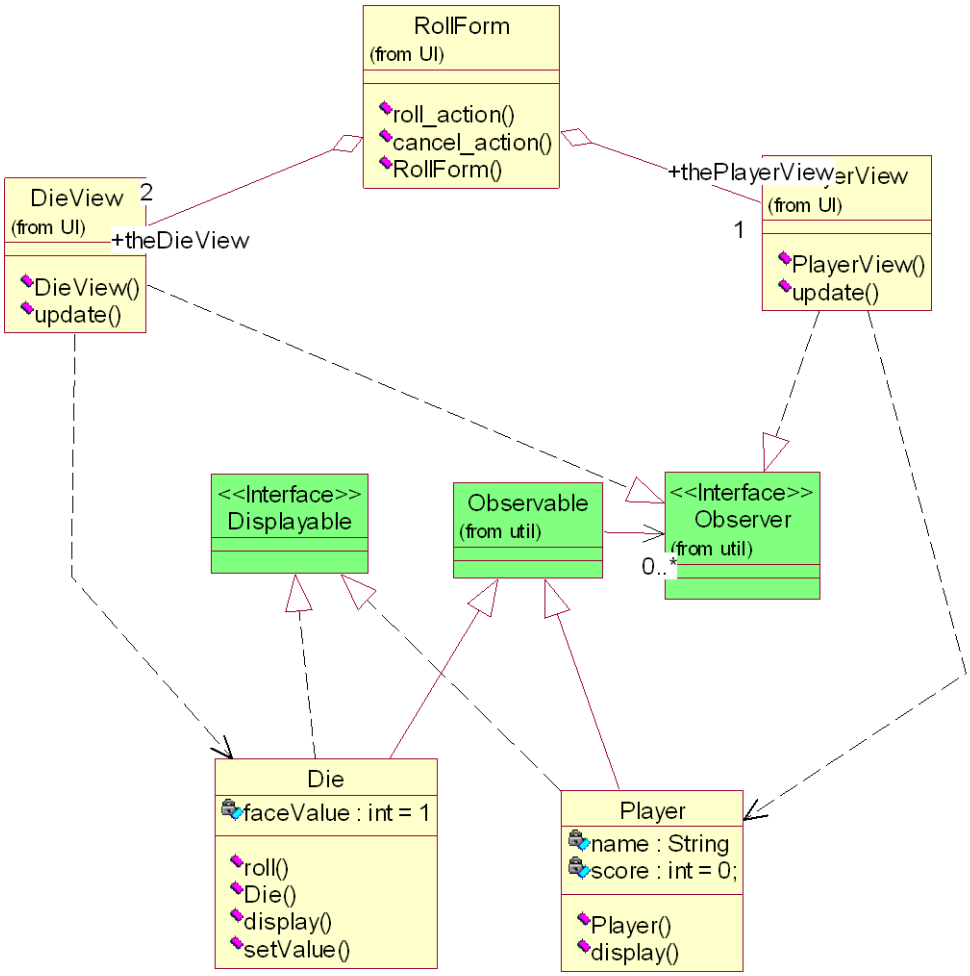
3-2-2 le pattern observer

Un objet (le sujet) est lié à plusieurs objets (les observateurs ou observers) pas nécessairement connus à la création du programme. Toute modification est automatiquement répercutée (notifiée) à tous les observateurs.

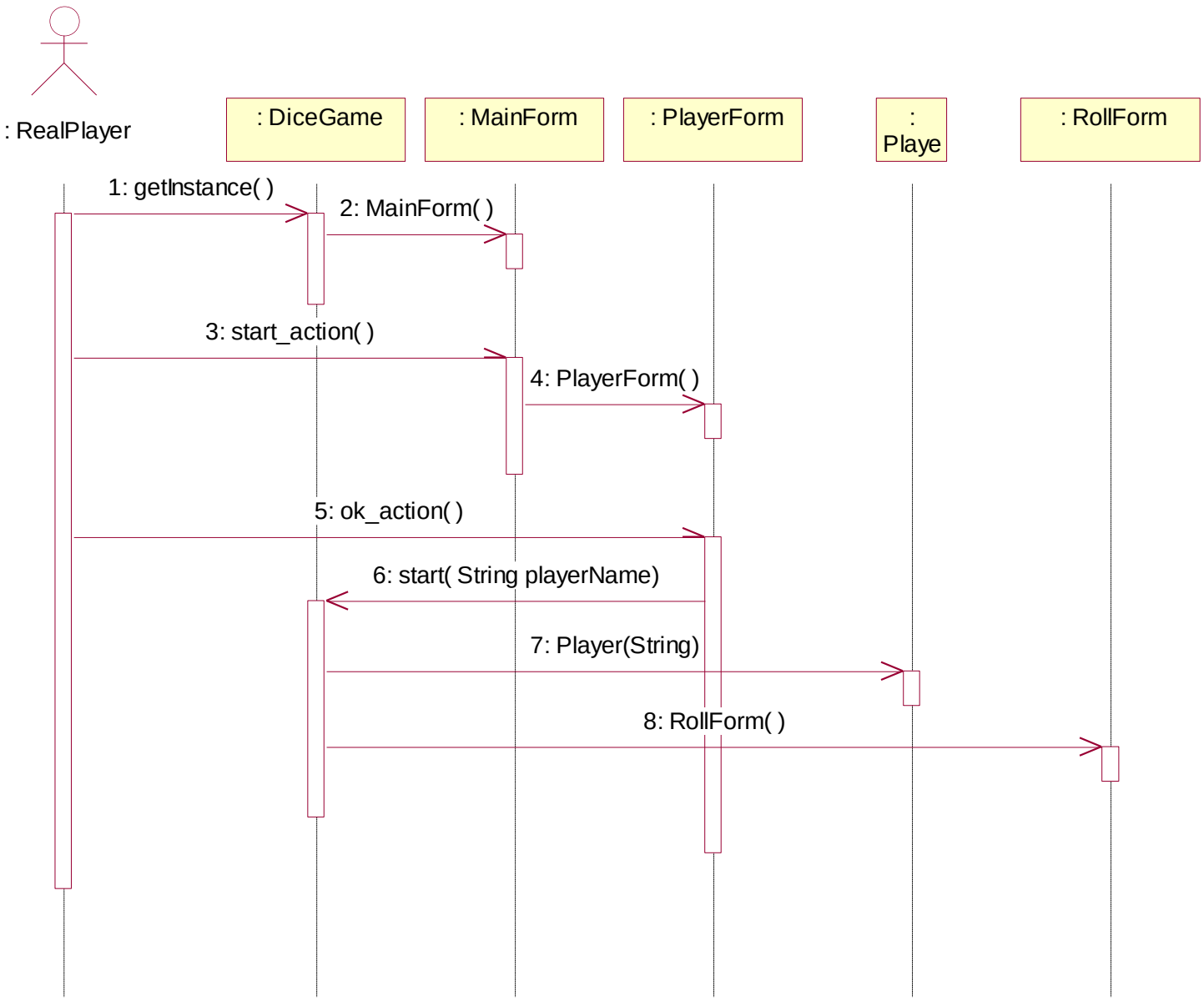
Ici, les dés et les joueurs sont les sujets (classe Observable) et les vues sur les dés et sur les joueurs sont les observateurs (interface Observer).

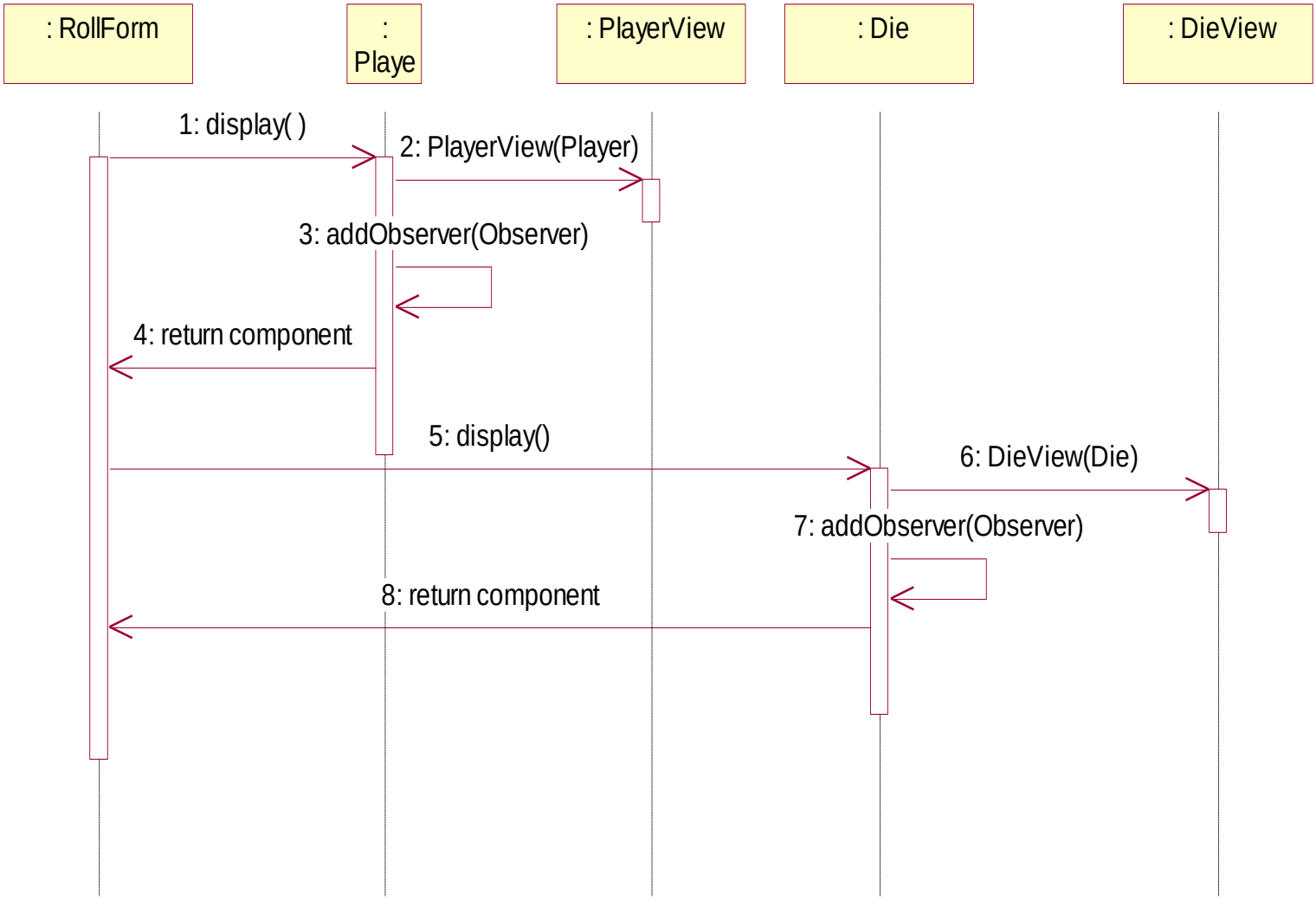


3-2-3 Diagramme de classe



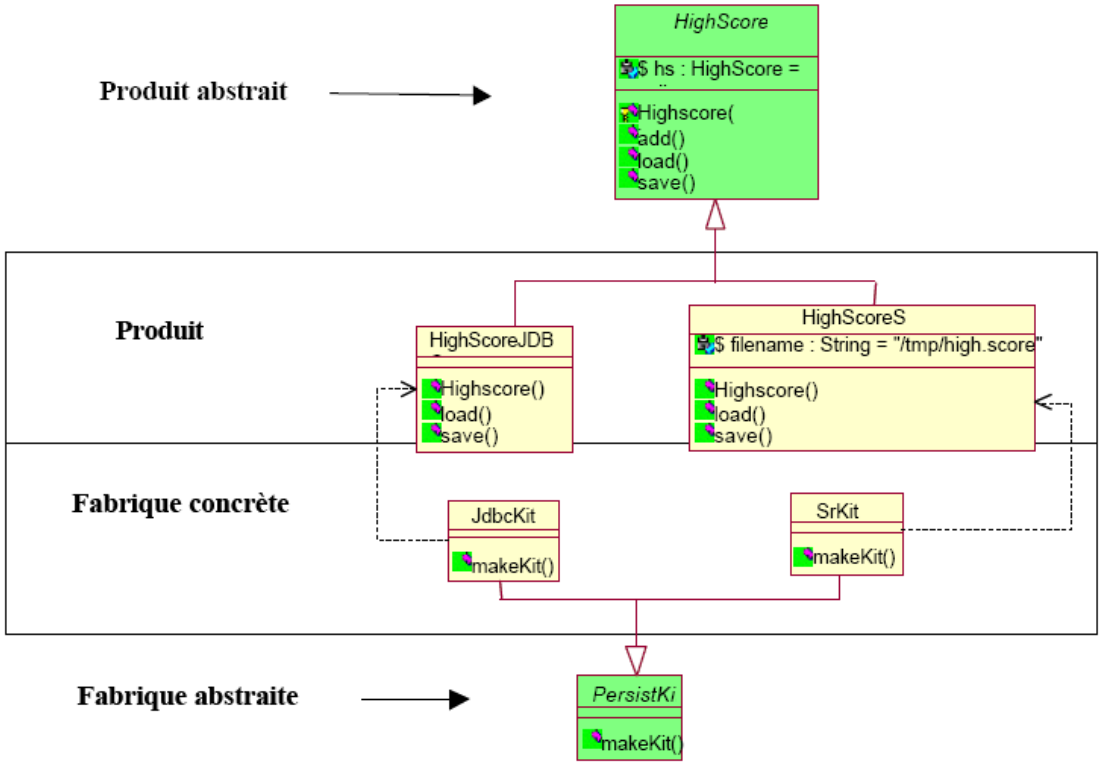
3-2-4 Diagramme de séquence





3-3 Le niveau persistance Persist

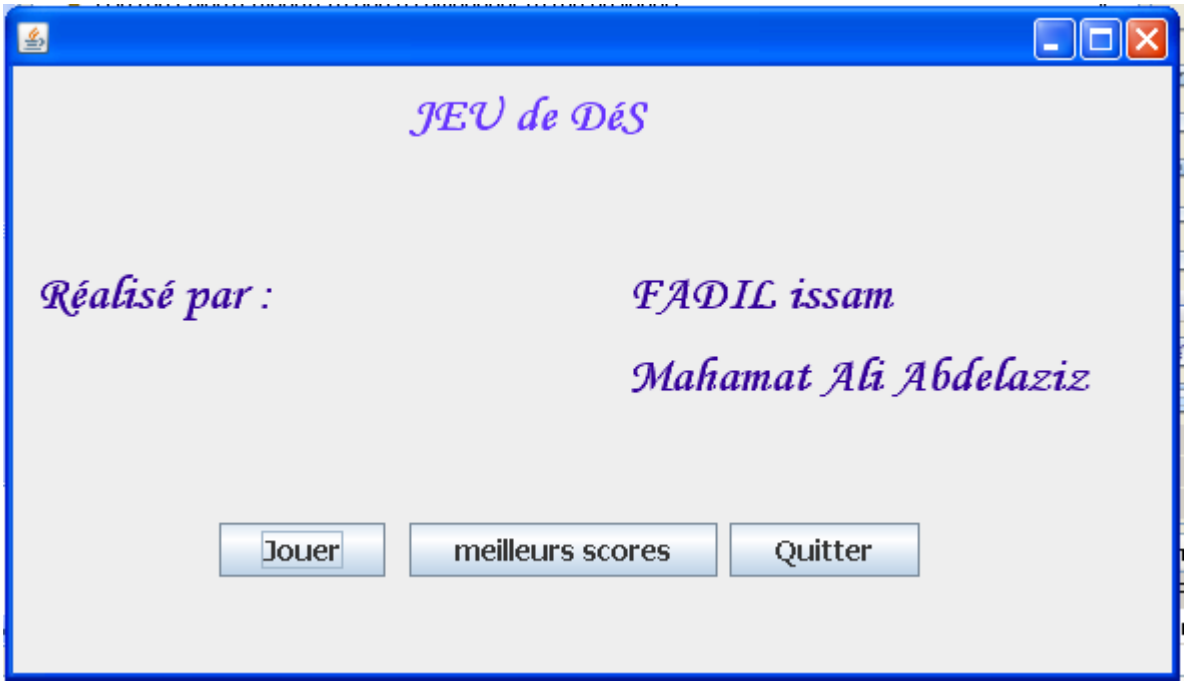
Il contient les classes techniques de persistance. L'objectif est d'assurer l'indépendance Core/Persist afin de pouvoir utiliser plusieurs types de persistance. Par exemple, la sérialisation (persistance d'objets liés dans un fichier) et l'utilisation d'une base de données relationnelle (via JDBC). Pour cela, on fait appel au pattern Factory. Ce pattern sert quand une classe peut créer des objets (ProduitConcret) de différentes classes. Une interface unique (Fabrication) est implantée par différentes classes concrètes pour chaque type d'objet à créer (FabricationConcrète). La classe qui veut créer les objets peut ne travailler qu'avec les interfaces.



3-4 Conception du niveau interface utilisateur

Il faut définir les fenêtres graphiques (‘forms’) contenant éventuellement les panneaux vues. Toutes les forms héritent de la classe awt Frame.

Fenêtre de démarrage



Le bouton jouer pour commencer une partie de jeu.
Le bouton meilleurs scores pour voir une liste des meilleurs scores obtenus.
Le bouton quitter pour quitter l'application.

Fenêtre d'identification

Fenêtre de jeu

Score : affiche le score obtenu
N° d'essais : nombre d'essai jouer, il s'arrête à 10.
Dé 1 : le résultat de premier dé.
Dé 2 : le résultat de deuxième dé.

Conclusion

L'analyse des besoins a utilisé
· cas d'utilisations + descriptions,
· diagramme d 'activités,
· prototypage de l'interface utilisateur.

L'analyse a utilisé
· pour la dynamique : les diagrammes de collaborations, séquences, états,
· pour la statique : les diagrammes de classes.

La conception a utilisé :
· le pattern architectural Layer
· les diagrammes de paquetages, de composants, de déploiement,
· les design patterns Singleton, Observer, Factory,
· des classes techniques pour la persistance et l'interface utilisateur.

La réalisation a utilisé les outils de génération et de rétro conception. Il est indispensable de mettre à jour la documentation d'analyse et de conception. Les tests ont utilisé les cas d'utilisation (couverture des fonctionnalités) et le diagramme d'activité (conformité). Bien entendu sur des grosses applications les démarches de test sont plus complexes (intégration, non régression, ...).

UML permet d'avoir plusieurs points de vue à chaque étape et de réfléchir en termes de couverture et de cohérence. Le travail est facilité avec un atelier UML.

UML permet de documenter les choix d'analyse et de conception. Grâce au processus de développement maîtrisé, le produit est conforme à ce qui était prévu au départ. Grâce aux patterns, le produit est évolutif (on peut facilement modifier l'interface ou la persistance). Grâce à Java, le produit est portable (système, SGBD).